0. Hello World with Function

Write a Ruby script that prints “Hello, Holberton! from Ruby!” using a function that accepts one argument String and print `Hello, Holberton! from <str>`.

- Function prototype: `say_hello(str)`

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ cat 0-main.rb
require_relative '0-hello_world_function'

say_hello("Ruby!")
```

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ ruby 0-main.rb
Hello, Holberton! from Ruby!
```

Step 1: Create the Ruby script file `0-hello_world_function.rb`

This file defines the `say_hello` function.

```ruby
# 0-hello_world_function.rb

def say_hello(str)
  puts "Hello, Holberton! from #{str}"
end
```

Step 2: Create the main script `0-main.rb`

This file requires the function from the first file and calls it with the given argument.

```ruby
# 0-main.rb

require_relative '0-hello_world_function'

say_hello("Ruby!")
```

Step 3: Test the Script

Run the script from the terminal to verify the output:

```
ruby 0-main.rb
```

Expected Output:

```
Hello, Holberton! from Ruby!
```

Why Two Files?

1. **File 1:** `0-hello_world_function.rb`
   - Contains the function definition.
   - This keeps your functions modular and reusable.
2. **File 2:** `0-main.rb`
   - Acts as the entry point.
   - This is where you call the function to execute it.

The Benefit

Separating the function and the main script makes your code cleaner and more maintainable, which aligns with best practices for larger projects.

What Does `require_relative` Do?

`require_relative` tells Ruby to load code from another file located **relative to the current file**. After including the file, all the methods, classes, or variables defined in that file become available to the script.